Standard Operating Protocol

DeepLabCut + SimBA Pipeline

Pose Estimation & Behavioral Classification
for Top-View Mouse Recordings (Single & Multi-Animal)

Borkar Lab
Miami University, Department of Psychology

Author: Siyi Li (Annemi)
Version: 1.0 | April 2026

Table of Contents

1. Purpose and Scope

This protocol provides step-by-step instructions for using DeepLabCut (DLC) for markerless pose estimation and SimBA (Simple Behavioral Analysis) for supervised behavioral classification. It is designed for researchers in the Borkar Lab who need to analyze top-view video recordings of mice (single-animal and multi-animal).

1.1 Target Behaviors

The current pipeline is configured to classify four behaviors:

- Freezing (cessation of all movement except respiration)
- Rearing (vertical extension of the body, forepaws off the floor)
- Tail rattling (rapid lateral oscillation of the tail)
- Grooming (repetitive self-directed movements of the forepaws to face/body)

1.2 Pipeline Overview

The complete workflow is:

1. **Video acquisition** → Record top-view videos (not covered in this protocol)
2. **DeepLabCut** → Extract frames, label body parts, train network, analyze videos
3. **SimBA** → Import pose data, extract features, annotate behaviors, train classifiers, run inference
4. **Export & statistics** → Export frame-by-frame classifications, generate summary statistics

NOTE	Sections 4.1“4.11 cover the single-animal workflow. For multi-animal (social interaction) recordings, see Section 5: Multi-Animal DeepLabCut.

2. System Requirements

2.1 Hardware

Component	Minimum	Recommended
GPU	NVIDIA with 4 GB VRAM	NVIDIA RTX 3060+ (8 GB+)
RAM	16 GB	32 GB+
Storage	100 GB free (SSD)	500 GB+ SSD
CPU	Intel i5 / AMD Ryzen 5	Intel i7+ / Ryzen 7+
OS	Windows 10 64-bit	Windows 10/11 64-bit

TIP	DLC can train on CPU but will be 5-10x slower. For a lab with limited GPU access, consider training overnight or using Google Colab (free tier provides a T4 GPU).

2.2 Software Stack

Software	Version	Notes
Anaconda / Miniconda	Latest	Package & environment manager
CUDA Toolkit	11.8	Must match your TensorFlow/PyTorch build
cuDNN	8.6+	Bundled if installing via conda
DeepLabCut	2.3.x+	Install via pip inside conda env
SimBA	1.x (latest)	Install via pip; separate conda env recommended
FFmpeg	Latest	For video conversion; conda install ffmpeg

3. Environment Setup

3.1 Conda Environment for DeepLabCut

Open Anaconda Prompt (Run as Administrator is NOT needed) and run:

```
conda create -n dlc python=3.9 -y
conda activate dlc
pip install deeplabcut[tf]
# OR for GPU (strongly recommended):
pip install deeplabcut[gui,tf]
```

Verify GPU detection:

```
python -c "import tensorflow as tf; print(tf.config.list_physical_devices('GPU'))"
```

You should see a list containing at least one GPU device. If empty, see Section 8 (Troubleshooting).

3.2 Conda Environment for SimBA

SimBA should be installed in a SEPARATE conda environment to avoid dependency conflicts:

```
conda create -n simba python=3.6 -y
conda activate simba
pip install simba-uw-tf-dev
```

WARNING	Do NOT install DLC and SimBA in the same environment. Their TensorFlow / scikit-learn version requirements often conflict. Always activate the correct environment before use.

3.3 Verifying Installation

Run the following quick checks in their respective environments:

```
# In dlc environment:
conda activate dlc
python -c "import deeplabcut; print(deeplabcut.__version__)"

# In simba environment:
conda activate simba
python -c "import simba; print('SimBA OK')"
```

4. DeepLabCut Workflow (Single Animal)

All DLC commands below assume you have activated the dlc conda environment. You can run DLC either through its Python API or through the GUI. This protocol shows the Python API; the GUI equivalents are noted where relevant.

4.1 Creating a New Project

Step 1. Create the project directory structure:

```
import deeplabcut

configPath = deeplabcut.create_new_project(
    'single_mice_topview',        # project name
    'YourName',                    # experimenter
    [r'C:\videos\video1.avi'],   # initial video(s)
    working_directory=r'C:\DLC_Projects',
    copy_videos=True               # copies video into project
)
print(configPath)  # save this path!
```

This creates the following directory tree:

```
single_mice_topview-YourName-YYYY-MM-DD/
├── config.yaml              # master config
├── dlc-models/              # trained models go here
├── labeled-data/            # extracted + labeled frames
├── training-datasets/       # generated training sets
└── videos/                  # source videos
```

Step 2. Edit `config.yaml` to define body parts. Open the file in a text editor and modify:

```
bodyparts:
  - snout
  - left_ear
  - right_ear
  - center_spine
  - left_forepaw
  - right_forepaw
  - left_hindpaw
  - right_hindpaw
  - tail_base
  - tail_mid
  - tail_tip
```

TIP	Choose body parts that are visible from the top-down view and relevant to your target behaviors. Tail points are critical for tail-rattling detection. Paw points help with freezing and grooming.

Step 3. Also set the skeleton (connections for visualization) and the number of frames to extract:

```
skeleton:
  - [snout, center_spine]
  - [center_spine, tail_base]
  - [tail_base, tail_mid]
  - [tail_mid, tail_tip]
```

```
    - [left_ear, snout]
    - [right_ear, snout]
    - [center_spine, left_forepaw]
    - [center_spine, right_forepaw]
    - [center_spine, left_hindpaw]
    - [center_spine, right_hindpaw]

numframes2pick: 20    # per video
```

4.2 Adding Videos to an Existing Project

To add additional videos after project creation:

```
deeplabcut.add_new_videos(
    configPath,
    [r'C:\videos\video2.avi', r'C:\videos\video3.avi'],
    copy_videos=True
)
```

NOTE	Only add videos with the same camera angle, resolution, and experimental setup. Mixing camera perspectives will degrade model performance.

4.3 Extracting Frames

DLC selects representative frames from your videos for manual labeling. The default k-means algorithm picks frames that are maximally diverse:

```
deeplabcut.extract_frames(
    configPath,
    mode='automatic',    # 'automatic' (k-means) or 'manual'
    algo='kmeans',
    userfeedback=False   # set True to approve each frame
)
```

Extracted frames are saved to `labeled-data/<videoname>/` as .png files.

4.4 Labeling Frames

This is the most time-intensive step. Quality here directly determines model accuracy.

Step 4. Launch the labeling GUI:

```
deeplabcut.label_frames(configPath)
```

Step 5. In the GUI that opens, navigate to each frame and click on each body part in the order listed in your config.yaml.

4.4.1 Labeling Best Practices

- Always label the center of the body part, not the edge.
- If a body part is occluded (not visible), do NOT place a label. Leave it empty. DLC handles missing labels.
- Zoom in for precision. Use the scroll wheel or the magnifying glass tool.
- Be consistent across all frames. If you define snout as the tip of the nose, always place it at the tip.
- Label at least 100–200 frames total across your videos for a robust model. 20 frames per video with 5–10 videos is a good starting point.

Step 6. When done, click **Save** in the GUI. Labels are stored as `CollectedData_<experimenter>.h5` and `.csv` in each video's labeled-data folder.

WARNING	NEVER rename or move video files after labeling. DLC maps labels to videos by absolute path. If you must reorganize, update the paths in config.yaml manually.

4.5 Creating the Training Dataset

This step merges all labeled frames into a single training set and applies data augmentation:

```
deeplabcut.create_training_dataset(
    configPath,
    net_type='resnet_50',        # backbone (resnet_50 recommended)
    augmenter_type='imgaug'      # data augmentation library
)
```

This generates a `training-datasets/` folder and a `pose_cfg.yaml` inside `dlc-models/iteration-0/.../train/`.

TIP	resnet_50 is the best balance of accuracy and training speed. resnet_101 or efficientnet may provide marginal improvements at much higher computational cost.

4.6 Training the Network

This is the computationally heavy step. On a GPU, training typically takes 2–6 hours. On CPU, expect 12–48+ hours.

```
deeplabcut.train_network(
    configPath,
    shuffle=1,
    displayiters=1000,
    saveiters=50000,
    maxiters=200000,    # increase if loss is still decreasing
    allow_growth=True   # prevents TF from allocating all GPU memory
)
```

4.6.1 Monitoring Training

- Watch the **training loss** printed every displayiters steps. It should decrease over time.
- A final training loss below `0.005` (or train error < 3 px) is generally good for top-view mouse tracking.
- Training can be interrupted (Ctrl+C) and resumed from the last snapshot.
- To resume: simply re-run `train_network()` with the same arguments; DLC auto-detects the latest snapshot.

WARNING	If you see "ResourceExhaustedError" (OOM), reduce the batch_size in pose_cfg.yaml (default 1). If using a shared machine, ensure no other GPU-heavy processes are running.

4.7 Evaluating the Network

Evaluate model performance on the labeled frames:

```
deeplabcut.evaluate_network(
    configPath,
    plotting=True    # generates labeled evaluation images
)
```

This prints two key metrics:

Metric	Meaning	Target
Train error	Average pixel distance on training frames	< 3 px (excellent), < 5 px (acceptable)
Test error	Average pixel distance on held-out frames	< 8 px (good), < 5 px (excellent)

Check the evaluation images in `evaluation-results/`. If labels are systematically off on certain body parts, consider relabeling those and retraining.

4.8 Analyzing New Videos

Run the trained model on your experimental videos to generate pose estimates:

```
deeplabcut.analyze_videos(
    configPath,
    [r'C:\videos\experiment_day1.avi'],
    shuffle=1,
    save_as_csv=True,
    videotype='.avi'
)
```

Output: a `.h5` and `.csv` file per video, containing (x, y, likelihood) for each body part on every frame.

4.9 Creating Labeled Videos (Visualization)

Generate videos with pose overlays for visual QC:

```
deeplabcut.create_labeled_video(
    configPath,
    [r'C:\videos\experiment_day1.avi'],
    draw_skeleton=True,
    filtered=True       # use filtered output if available
)
```

Always review labeled videos before proceeding to SimBA. Look for: points jumping to wrong locations, tracking loss during fast movement, and consistently mislabeled body parts.

4.10 Filtering Predictions (Optional but Recommended)

DLC provides post-hoc filtering to smooth jittery predictions:

```
deeplabcut.filterpredictions(
    configPath,
    [r'C:\videos\experiment_day1.avi'],
    filtertype='median',   # or 'arima', 'spline'
    windowlength=5
)
```

Filtering produces a new file with the suffix _filtered. SimBA can import either filtered or unfiltered data.

4.11 Refining Labels (Outlier Correction)

If evaluation reveals poor predictions on certain postures, extract those frames and relabel:

```
deeplabcut.extract_outlier_frames(
    configPath,
    [r'C:\videos\experiment_day1.avi'],
    outlieralgorithm='jump',    # detect sudden jumps
    epsilon=20                   # pixel threshold
)
```

```
deeplabcut.refine_labels(configPath)

# After correcting labels in the GUI:
deeplabcut.merge_datasets(configPath)
# Then re-create training dataset and retrain
```

5. Multi-Animal DeepLabCut (maDLC)

When your experiment involves multiple mice in the same arena (e.g., social interaction, resident–intruder, tube tests), you need the multi-animal variant of DLC. maDLC adds identity-aware tracking so that each animal's pose is tracked separately and consistently across frames.

5.1 When to Use maDLC vs. Standard DLC

Use Standard DLC (Section 4)	Use maDLC (this section)
One animal per video	Two or more animals per video
Animals never overlap	Animals may interact, cross, or occlude
No need to distinguish individuals	Need per-animal trajectories

5.2 Project Creation for maDLC

Step 7. Create a multi-animal project by specifying the number of individuals:

```
import deeplabcut

configPath = deeplabcut.create_new_project(
    'social_interaction',
    'YourName',
    [r'C:\videos\pair_video1.avi'],
    working_directory=r'C:\DLC_Projects',
    copy_videos=True,
    multianimal=True     # <-- key flag
)
```

Step 8. Open `config.yaml` and configure multi-animal settings:

```
multianimalproject: true
individuals:
  - mouse1
  - mouse2

uniquebodyparts: []           # parts present once (e.g., food pellet)
multianimalbodyparts:         # parts present on EACH animal
  - snout
  - left_ear
  - right_ear
  - center_spine
  - left_forepaw
  - right_forepaw
  - left_hindpaw
  - right_hindpaw
  - tail_base
  - tail_mid
  - tail_tip
```

TIP	uniquebodyparts is for landmarks that appear exactly once in the scene (e.g., an arena corner marker or a food source). Leave it empty if there are no such points.

5.3 Labeling for Multi-Animal

The labeling workflow is similar to single-animal, but you must assign each label to a specific individual:

Step 9. Extract frames as usual:

```
deeplabcut.extract_frames(configPath, mode='automatic', algo='kmeans')
```

Step 10. Launch the labeling GUI:

```
deeplabcut.label_frames(configPath)
```

5.3.1 Multi-Animal Labeling Rules

- The GUI shows a dropdown or tab for each individual (mouse1, mouse2). Select the correct animal before placing labels.
- **Consistency matters:** If mouse1 is the lighter-colored mouse in one frame, it must be mouse1 in ALL frames. Use fur color, ear marks, or other distinguishing features.
- If two animals are so close that you cannot distinguish their body parts, label what you can see and leave ambiguous points unlabeled.
- Label at least 150–300 frames total for multi-animal projects (more than single-animal because the task is harder).

WARNING	Identity swaps during labeling are the #1 source of poor maDLC performance. Double-check every frame's identity assignment. If you are unsure which mouse is which, skip that frame rather than guessing.

5.4 Training and Inference (maDLC)

Training uses the same commands as single-animal DLC but with additional parameters for the multi-animal assembly step:

Step 11. Create the training dataset:

```
deeplabcut.create_multianimaltraining_dataset(configPath)
```

Step 12. Train the network (same as single-animal):

```
deeplabcut.train_network(configPath, maxiters=200000, allow_growth=True)
```

Step 13. Evaluate:

```
deeplabcut.evaluate_multianimal_crossvalidate(
    configPath,
    Shuffles=[1],
    edgewisecondition=True,
    leastbpts=3,
    init_points=20,
    n_iter=50
)
```

This step performs cross-validation and optimizes the parameters for the **assembly** step (grouping detected body parts into individual animals).

Step 14. Analyze videos:

```
deeplabcut.analyze_videos(
    configPath,
    [r'C:\videos\pair_video1.avi'],
    auto_track=True     # runs assembly + tracking automatically
)
```

5.5 Identity Tracking and Stitching

After pose estimation, maDLC must assign detections to consistent animal IDs across frames. This is the assembly + stitching pipeline:

```
# If auto_track was not used in analyze_videos, run manually:
deeplabcut.convert_detections2tracklets(
    configPath,
    [r'C:\videos\pair_video1.avi'],
    shuffle=1,
    videotype='.avi',
    track_method='ellipse'    # or 'box', 'skeleton'
)

deeplabcut.stitch_tracklets(
    configPath,
    [r'C:\videos\pair_video1.avi'],
    shuffle=1,
    track_method='ellipse'
)
```

Track Method	How It Works	Best For
ellipse	Fits ellipses to detected body parts and tracks ellipse centroids	General use; good default

box	Uses bounding boxes around body part clusters	Animals rarely cross paths
skeleton	Uses skeleton geometry for matching	Animals with very different sizes

Create labeled videos to verify tracking identity assignments:

```
deeplabcut.create_labeled_video(
    configPath,
    [r'C:\videos\pair_video1.avi'],
    color_by='individual',    # color-codes each animal
    draw_skeleton=True
)
```

TIP	If identity swaps occur (colors switch between animals mid-video), try a different track_method or re-run stitch_tracklets with adjusted parameters. Persistent swaps usually indicate insufficient or identity-inconsistent labeled frames.

The final output CSV will have columns organized by individual (e.g., `mouse1_snout_x, mouse1_snout_y, mouse2_snout_x, ...`). This format is directly importable into SimBA for multi-animal behavioral classification.

6. SimBA Workflow

SimBA takes the pose estimation CSV output from DLC and builds supervised classifiers for behaviors of interest. Switch to the simba conda environment before proceeding.

NOTE	SimBA can be run via the GUI or command line. This protocol covers the GUI workflow, which is more intuitive for first-time users. CLI equivalents are noted where applicable.

6.1 Creating a SimBA Project

Step 15. Launch SimBA:

```
conda activate simba
simba
```

Step 16. In the GUI, go to **File > Create a new project**.

Step 17. Fill in the project details:

- **Project name:** descriptive name (e.g., fear_conditioning_2026)
- **Project directory:** choose a location with ample disk space
- **Pose estimation body part configuration:** select the body part schema that matches your DLC config (e.g., 11 body parts for our top-view setup)

TIP	If the pre-built schemas do not match your body part list, choose "User-defined" and enter your body parts in the exact same order as your DLC config.yaml.

Step 18. Under **SML settings**, define your classifiers:

- Classifier 1: Freezing
- Classifier 2: Rearing
- Classifier 3: Tail_rattling
- Classifier 4: Grooming

Step 19. Click **Create Project**. SimBA generates its own directory tree.

6.2 Importing DLC Tracking Data

Step 20. In the SimBA GUI, go to **File > Load project > Load project.ini** and open your project.

Step 21. Navigate to **Process tracking data > Import DLC tracking data**.

Step 22. Select the folder containing your DLC output CSV files (`*.csv` from analyze_videos).

Step 23. Also import the corresponding video files when prompted.

WARNING	Make sure each CSV file name matches its video file name (e.g., video1.csv pairs with video1.avi). SimBA matches files by name.

6.3 Correcting Outliers & Interpolation

SimBA provides built-in tools to clean noisy pose data before feature extraction:

Step 24. Go to **Process tracking data > Outlier correction**.

Step 25. Set outlier criteria. Recommended defaults:

- **Criterion:** mean + 2 SD (body-part displacement between frames)
- **Anchor body parts:** center_spine, tail_base (stable points)

Step 26. Run the outlier correction on all imported CSVs.

6.4 Feature Extraction

SimBA computes hundreds of geometric and temporal features from the pose data (distances, angles, velocities, areas, etc.):

Step 27. Go to **Extract features**.

Step 28. Click **Extract features**. This may take several minutes per video.

Output: feature CSV files saved to `project_folder/csv/features_extracted/`.

6.5 Annotating Behaviors (Ground Truth)

This is where you manually label the presence/absence of each behavior, frame by frame. This is the training data for the classifiers.

Step 29. Go to **Label behavior > Select video**.

Step 30. The annotation GUI will display the video. For each frame, mark the behaviors present using the checkboxes or keyboard shortcuts.

6.5.1 Annotation Best Practices

- Annotate at least 3–5 videos thoroughly (every frame) per behavior.
- Include videos where the target behavior is both present and absent.
- Be strict about onset/offset: mark the exact frame where the behavior starts and stops.

- Use a consistent definition. For example, define freezing as > 0.5 seconds of no visible movement.
- Save frequently. Annotations are stored per-video.

Step 31. After annotating, go to **Label behavior > Append annotations to features** to merge ground truth with extracted features.

6.6 Training Classifiers

Step 32. Navigate to **Train Machine Model**.

Step 33. Configure training parameters for each classifier:

Parameter	Recommended Value	Notes
Algorithm	Random Forest	Default; works well
n_estimators	2000	More trees = better (slower)
Hyperparameter search	Yes	Recommended for first run
Under-sample setting	Custom (by behavior)	Use if class imbalance is severe
Train/test split	80/20	Standard

Step 34. Click Train. SimBA trains one Random Forest classifier per behavior and outputs evaluation metrics (precision, recall, F1).

6.6.1 Evaluating Classifier Performance

After training, review the following outputs:

- **Classification report:** Precision, recall, and F1 per behavior. Aim for F1 > 0.80.
- **Confusion matrix:** Look for off-diagonal entries to understand misclassifications.
- **Feature importance:** Identifies which pose features drive classification. Useful for debugging.

TIP	If performance is low for a specific behavior, add more annotated examples of that behavior. Class imbalance is the #1 cause of poor recall.

6.7 Running Classifiers on New Data

Apply trained classifiers to analyze unannotated videos:

Step 35. Go to **Run Machine Model**.

Step 36. Select the classifier model files (.sav format) and set the discrimination threshold.

Threshold Setting	Effect	When to Use
Low (0.3–0.4)	More sensitive (fewer misses)	When false negatives are costly
Medium (0.5)	Balanced	Default starting point
High (0.6–0.8)	More specific (fewer false positives)	When precision matters more

Step 37. Click **Run**. Output CSVs with frame-level behavior predictions are saved to `project_folder/csv/machine_results/`.

6.8 Validation & Visualization

SimBA provides several visualization tools to verify classifier results:

- **Gantt plots:** Timeline bars showing behavior bouts across the video. Found in Visualization > Gantt.
- **Probability plots:** Frame-by-frame classifier confidence. Helps identify threshold tuning needs.
- **Validation videos:** Generate videos with behavior labels overlaid on the original recording.
- **Data tables:** Export summary statistics (bout count, total duration, mean bout length) per video per behavior.

TIP	Always review validation videos on a random subset before trusting batch results. Manual spot-checking catches systematic errors that summary stats cannot reveal.

7. Data Management

7.1 Directory Structure

Maintain the following top-level organization on the lab data drive:

```
D:\BehaviorAnalysis\
├── raw_videos/              # original recordings (NEVER modify)
├── DLC_Projects/
│   └── single_mice_topview-<experimenter>-<date>/
├── SimBA_Projects/
│   └── <project_name>/
├── results/                 # exported summary stats & figures
└── archive/                 # old models & deprecated configs
```

7.2 File Naming Convention

Use the following pattern for all video files:

```
<subjectID>_<condition>_<date>_<trial>.<ext>
Example: mouse042_FC_20260315_trial1.avi
```

- No spaces in file names. Use underscores.
- Keep names short but descriptive.
- Date format: YYYYMMDD.

7.3 Backup Strategy

- **Raw videos:** Back up to an external drive or lab NAS immediately after recording. Keep at least 2 copies.
- **DLC labeled data:** Back up the labeled-data/ folder after every labeling session. This represents hours of manual work.
- **Trained models:** Back up the dlc-models/ folder. Re-training is expensive.
- **SimBA annotations:** Back up the csv/targets_inserted/ folder after annotation sessions.
- **config.yaml:** Version-control this file (e.g., keep dated copies). It is the single source of truth for the DLC project.

8. Exporting Results for Statistical Analysis

SimBA's machine_results CSV files contain frame-by-frame predictions. To extract summary statistics:

Step 38. In SimBA, go to **Analysis > Analyze machine predictions**.

Step 39. Configure the output:

- Total duration (seconds) of each behavior per video
- Number of bouts per behavior per video
- Mean and median bout duration
- Latency to first bout

Step 40. Export to CSV. Files appear in `project_folder/logs/`.

These CSVs can be imported directly into R, Python (pandas), SPSS, or Prism for group-level statistical analysis.

TIP	For publication-quality analyses, also report the classifier performance (F1, precision, recall) alongside behavioral results. Reviewers increasingly expect this transparency.

9. Troubleshooting

9.1 GPU / CUDA Issues

Symptom	Solution
TF does not detect GPU	Verify CUDA version matches TF version (check TF compatibility matrix). Run: nvidia-smi and compare CUDA version. Reinstall cudatoolkit via conda if needed.
ResourceExhaustedError (OOM)	Reduce batch_size in pose_cfg.yaml to 1. Close other GPU applications. Reduce global_scale in pose_cfg.yaml.
CUDNN_STATUS_INTERNAL_ERROR	Add allow_growth=True to train_network() call. Or set TF_FORCE_GPU_ALLOW_GROWTH=true as environment variable.
Training runs on CPU despite GPU present	Reinstall tensorflow-gpu (not tensorflow) in your conda env. Confirm with: python -c "import tensorflow as tf; print(tf.test.is_gpu_available())"

9.2 DeepLabCut Common Errors

Error	Fix
"No data found" during training	Re-run create_training_dataset(). Ensure labeled .h5 files exist in labeled-data/ folders.
Labeling GUI does not open	Install wxPython: pip install wxPython. On Windows, try: conda install -c conda-forge wxpython.
Video codec error during analysis	Convert video to .avi (MJPEG) or .mp4 (H.264) using FFmpeg: ffmpeg -i input.ext -c:v mjpeg output.avi
Predictions are wildly off / jittery	Usually indicates insufficient or inconsistent labels. Add more labeled frames, especially frames showing the problematic postures. Retrain.
"Snapshot not found" on resume	Ensure the snapshot path in pose_cfg.yaml matches actual files in the train/ directory. Fix the init_weights path manually.

9.3 SimBA Common Issues

Issue	Solution
Body part mismatch on import	Ensure the body part names and order in SimBA project match the DLC config.yaml EXACTLY. Even capitalization matters.
Feature extraction fails / NaN values	Usually caused by missing pose data (e.g., all-zero frames). Run outlier correction first. Check DLC output for frames with 0 likelihood.
Very low F1 after training	Common causes: (1) too few annotated examples, (2) severe class imbalance, (3) vague behavior definitions. Add more annotations and use under-sampling.
SimBA GUI freezes during annotation	Reduce video resolution or downsample framerate before

	import. SimBA loads entire videos into memory.
Pickle version error loading .sav model	scikit-learn version mismatch. Ensure the same scikit-learn version is used for training and inference. Pin versions in your environment.

10. Quick Reference: DLC Commands

Copy-paste reference for the most common operations:

Operation	Command
Create project	`deeplabcut.create_new_project(name, exp, videos)`
Extract frames	`deeplabcut.extract_frames(config, mode='automatic')`
Label frames	`deeplabcut.label_frames(config)`
Create training set	`deeplabcut.create_training_dataset(config)`
Train network	`deeplabcut.train_network(config, maxiters=200000)`
Evaluate	`deeplabcut.evaluate_network(config, plotting=True)`
Analyze videos	`deeplabcut.analyze_videos(config, videos, save_as_csv=True)`
Filter predictions	`deeplabcut.filterpredictions(config, videos)`
Labeled video	`deeplabcut.create_labeled_video(config, videos)`
Outlier extraction	`deeplabcut.extract_outlier_frames(config, videos)`
Refine labels	`deeplabcut.refine_labels(config)`
Merge refined	`deeplabcut.merge_datasets(config)`

Appendix A: Recommended Body Part Scheme

The following 11-point scheme is optimized for single-mouse top-view analysis of freezing, rearing, grooming, and tail rattling:

#	Body Part	Landmark	Relevance
1	`snout`	Tip of nose	Heading direction; grooming detection
2	`left_ear`	Center of left pinna	Head orientation
3	`right_ear`	Center of right pinna	Head orientation
4	`center_spine`	Midpoint of dorsal spine	Body center; velocity/distance anchor
5	`left_forepaw`	Wrist joint (visible)	Grooming; freezing (paw immobility)
6	`right_forepaw`	Wrist joint (visible)	Grooming; freezing (paw immobility)
7	`left_hindpaw`	Ankle/heel (visible)	Rearing (hindpaw planted); freezing
8	`right_hindpaw`	Ankle/heel (visible)	Rearing (hindpaw planted); freezing
9	`tail_base`	Base of tail at body	Tail rattling detection; body length
10	`tail_mid`	Midpoint of tail	Tail curvature and movement
11	`tail_tip`	Tip of tail	Tail rattling amplitude

Appendix B: Key config.yaml Parameters

Parameter	Default	Description
`numframes2pick`	`20`	Number of frames extracted per video for labeling
`TrainingFraction`	`[0.95]`	Fraction of labeled data used for training (rest for test)
`iteration`	`0`	Current refinement iteration; increments with merge_datasets
`default_net_type`	`resnet_50`	Neural network backbone
`default_augmenter`	`imgaug`	Data augmentation strategy
`pcutoff`	`0.6`	Likelihood cutoff for plotting and filtering
`batch_size`	`1`	In pose_cfg.yaml; increase if VRAM allows
`global_scale`	`0.8`	In pose_cfg.yaml; scale factor for input images
`display_iters`	`1000`	How often to print training loss
`save_iters`	`50000`	How often to save a model snapshot

Appendix C: Glossary

Term	Definition
DLC	DeepLabCut — a markerless pose estimation toolbox using deep learning
SimBA	Simple Behavioral Analysis — a supervised machine learning pipeline for classifying behaviors from pose data
ResNet	Residual Network — a deep CNN architecture used as the backbone for DLC pose estimation
pcutoff	Prediction confidence cutoff — body part predictions below this likelihood are considered unreliable
Bout	A continuous episode of a single behavior; defined by onset and offset frames
F1 Score	Harmonic mean of precision and recall; a single metric for classifier performance (0–1, higher is better)
Random Forest	An ensemble machine learning algorithm that SimBA uses by default for behavior classification
Outlier frame	A video frame where DLC predictions deviate significantly from expected body geometry
Feature extraction	The process of computing mathematical descriptors (distances, angles, velocities) from raw pose coordinates
Snapshot	A saved checkpoint of the DLC neural network during training